Exercises: Objects, Classes and Collections

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

1. Basic Stack Operations

You will be given an integer **N** representing the **number of elements to push onto a stack**, an integer **S** representing the **number of elements to pop from the stack** and finally an integer **X**, an element **that you should check whether is present in the stack**. If it is, print **true** on the console. If it's not, print the smallest element currently present in the stack.

Input

- On the first line, you will be given **N**, **S** and **X** separated by a single space.
- On the next line, you will be given a line of numbers **separated by one or more white spaces**.

Examples

Input	Output	Comments
5 2 13 1 13 45 32 4	true	We have to **push 5** elements. Then we **pop 2** of them. Finally, we have to check whether 13 is present in the stack. Since it is we print **true**.
4 1 666 420 69 13 666	13	Pop one element (666) and then check if 666 is present in the stack. It's not, so print the smallest element (13)

2. Maximum Element

You have an empty sequence and you will be given **N** commands. Each command is of the following types:

- **"1 X"** - **Push** the element **X** into the stack.
- **"2"** - **Delete** the element present at the top of the stack.
- **"3"** - **Print** the maximum element in the stack.

Input

- The first line of input contains an integer **N**, where **$1 \leq N \leq 10^5$**
- The next **N** lines contain commands. All commands will be valid and in the format described
- The element **X** will be in range **$1 \leq X \leq 10^9$**

Examples

Input	Output	Comments
9 1 97 2 1 20	26 91	9 commands Push 97 Pop an element Push 20

2 1 26 1 20 3 1 91 3		Pop an element Push 26 Push 20 Print the maximum element (26) Push 91 Print the maximum element (91)

3. Basic Queue Operations

You will be given an integer **N** representing the **number of elements to enqueue** (add), an integer **S** representing the **number of elements to dequeue** (remove/poll) from the queue and finally an integer **X**, an element that you should **check whether is present in the queue**. If it is print **true** on the console, if it's not **print the smallest element currently present in the queue**.

Examples

Input	Output	Comments
5 2 32 1 13 45 32 4	true	We have to **push 5** elements. Then we **pop 2** of them. Finally, we have to check whether 13 is present in the stack. Since it is we print **true**.
4 1 666 666 69 13 420	13	

4. Truck Tour

Suppose there is a circle. There are **N** petrol pumps on that circle. Petrol pumps are numbered 0 to N - 1 (both inclusive). You will get **N** on the first line.

On the next N lines, you will be given:

- the **amount of petrol** that particular petrol pump will give
- the **distance from that petrol pump** to the next petrol pump.

Initially, you have a tank of **infinite** capacity carrying **no** petrol. You can start the tour at **any** of the petrol pumps.

Considering that the truck will stop at **each of the petrol pumps,** calculate the **first point (**The smallest index of a petrol pump station**)** from where the truck will be able to complete a full circle. The truck will **move one kilometer for each liter of petrol**.

Input

- The first line will contain the value of **N**: **1 ≤ N ≤ 1000001**
- The next **N** lines will contain a pair of integers each, **i.e. the amount of petrol that petrol pump will give and the distance between that petrol pump and the next petrol pump**.
- **1 ≤ Amount of petrol, Distance ≤ 1000000000**

Examples

Input	Output
3 1 5 10 3 3 4	1

5. Balanced Parentheses

Given a sequence consisting of parentheses, determine **whether an expression is balanced**. A sequence of parentheses **is balanced if** every open parenthesis can be paired **uniquely** with a closed parenthesis that occurs after the former. Also, **the interval between them must be balanced**. You will be given three types of parentheses: (, {, and [.

{[()]} - This is a balanced parenthesis.

{[(])} - This is not a balanced parenthesis.

Input

- Each input consists of a single line, the sequence of parentheses.
- **1 ≤ Length of sequence ≤ 1000**
- Each character of the sequence will be one of the following: **{, }, (,), [,]**.

Examples

Input	Output
{[()]}	YES
{[(])}	NO
{{[[(())]]}}	YES

6. *Simple Text Editor

You are given an empty text. Your task is to implement **4 types of commands** related to manipulating the text:

- **"1 [string]"** - **appends** [string] to the end of the text
- **"2 [count]"** - **erases** the last [count] elements from the text
- **"3 [index]"** - **returns** the element at position [index] from the text
- **"4"** - **undoes** the last not-undone command of type 1 or 2 and returns the text to the state before that operation

Input

- The first line contains **N**, the number of operations, where **1 ≤ N ≤ 105**
- Each of the following **N** lines contains the name of the operation, followed by the command argument, if any, separated by space in the following format **"command argument"**.

- **The length of the text** will not exceed **1000000**
- All input characters are **English letters**
- It is **guaranteed** that the sequence of **input operation is possible to perform**

Output

- For each operation of type **"3"** print a **single line with the returned character of that operation**.

Examples

Input	Output	Comments
8 1 abc 3 3 2 3 1 xy 3 2 4 4 3 1	c y a	There are **8 operations**. Initially, the **text is empty**. Append "abc" Print third character Erase 3 characters Append "xy" Print second character Undo last command - text is now "" Undo last command - text is now "abc" Print first character

7. *Infix to Postfix

Mathematical expressions are **written in an infix notations**, for example "5 / (3 + 2)". However, this kind of notation is **not efficient for computer processing**, as you first need to evaluate the expression inside the brackets, so there is a lot of back and forth movement. A more suitable approach is to **convert it in the so-called postfix notations** (also called Reverse Polish Notation), in which the **expression is evaluated from left to right**, for example "3 2 + 5 /".

Implement an **algorithm that converts** the mathematical expression **from infix notation into a postfix notation**. Use the famous Shunting-yard algorithm.

Input

- You will **receive an expression on a single line, consisting of tokens**
- Tokens could be numbers 0-9, variables a-z, operators +, -, *, / and brackets (or)
- Each token is **separated by exactly one space**

Examples

Input	Output
5 / (3 + 2)	5 3 2 + /
1 + 2 + 3	1 2 + 3 +
7 + 13 / (12 - 4)	7 13 12 4 - / +
(3 + x) - y	3 x + y -

8. *Evaluate Expression

Use your previous solution to **convert a given expression from infix to postfix** notation and to **evaluate its result**.

Input

- You will **receive an expression on a single line, consisting of tokens**
- Each token will be **separated by exactly one space**.

Output

- **Print the result** of the expression.
- **Format** the output to the **second digit after the decimal separator**.

Examples

Input	Output
5 / (3 + 2)	1.00
1 + 2 + 3	6.00
7 + 13 / (12 - 4)	8.63

9. *The Stock Span Problem

The **stock span problem** is a financial problem where we have a **series of daily price quotes** for a stock and we need to **calculate the span of stock's price for all n days**. Span is defined as the number of consecutive days before the given day where the price of stock was less than or equal to price at the given day.

You can read about it here: http://www.geeksforgeeks.org/the-stock-span-problem/

Implement an efficient algorithm that **calculates the spans for a given n stock prices**.

Input

- On the **first line**, you will receive **n**, the **number of stock prices**.
- On the **next n lines**, you will get **all prices**.

Examples

Input	Output
7 9 7 4 5 4 6 8	1 1 1 2 1 4 6

Input	Output
5 1 2 3 4 5	1 2 3 4 5

Input	Output
5 5 4 3 2 1	1 1 1 1 1

Input	Output
4 3 2 1 3	1 1 1 4

10. Count Symbols

Write a program that reads some text from the console and counts the occurrences of each character in it. Print the results in **alphabetical** (lexicographical) order.

Examples

Input	Output
SoftUni rocks	: 1 time/s S: 1 time/s U: 1 time/s c: 1 time/s f: 1 time/s i: 1 time/s k: 1 time/s n: 1 time/s o: 2 time/s r: 1 time/s s: 1 time/s t: 1 time/s

11. Phonebook

Write a program that receives some info from the console about **people** and their **phone numbers**. Each **entry** has just **one name** and **one number**. If you receive a name that **already exists** in the phonebook, simply update its number.

After filling this simple phonebook, upon receiving the **command** "**`search`**", perform a search of a contact by name and print its details in format "**`{name} -> {number}`**".

In case the contact isn't found, print "**`Contact {name} does not exist.`**"

The output will end with a "Stop" command.

Examples

Input	Output
Nakov-0888080808 **search** Mariika Nakov Stop	Contact Mariika does not exist. Nakov -> 0888080808
Nakov-+359888001122 RoYaL(Ivan)-666 Gero-5559393 Simo-02/987665544	Simo -> 02/987665544 Contact simo does not exist. Contact RoYaL does not exist. RoYaL(Ivan) -> 666

search Simo simo RoYaL RoYaL(Ivan) Stop	

12. A Miner Task

You are given a sequence of strings, each on a new line. Every **odd** line on the console is representing a resource (e.g. Gold, Silver, Copper, and so on) and every **even** – quantity. Your task is to collect the resources and print them.

Print the resources and their quantities in format: {resource} –> {quantity}

Examples

Input	Output
Gold 155 Silver 10 Copper 17 stop	Gold -> 155 Silver -> 10 Copper -> 17

13. Hands of Cards

You are given a sequence of people and for every person the cards he draws from the deck. The input will be on separate lines in the format: **{personName}: {PT, PT, PT, … PT}**, where **P** (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A) is the power of the card and **T** (S, H, D, C) is the type. The input ends when a **"JOKER"** is drawn.

The name can contain any ASCII symbol except **':'**

A single person **cannot have more than one** card with the same power and type, if he draws such a card he discards it. The people are playing with **multiple decks**. Each card has a value that is calculated by the power multiplied by the type. Powers **2 to 10** have the same value and **J to A** are **11 to 14**. Types are mapped to multipliers the following way (**S -> 4, H-> 3, D -> 2, C -> 1**).

Finally print out the total value each player has in his hand in the format: **{personName}: {value}**

Examples

Input	Output
Pesho: 2C, 4H, 9H, AS, QS Slav: 3H, 10S, JC, KD, 5S, 10S Peshoslav: QH, QC, QS, QD Slav: 6H, 7S, KC, KD, 5S, 10C	Pesho: 167 Slav: 175 Peshoslav: 197

Peshoslav: QH, QC, JS, JD, JC Pesho: JD, JD, JD, JD, JD, JD JOKER	

14. * Population Counter

So many people! It's hard to count them all. But that's your job as a statistician. You get raw data for a given city and you need to aggregate it.

On each input line you'll be given data in format: **"city|country|population"**. There will be **no redundant whitespaces anywhere** in the input. A city-country pair will not be repeated. Aggregate the data **by country and by city** and print it on the console. For each country, print its **total population** and on separate lines the data for each of its cities. **Countries should be ordered by their total population in descending order** and within each country, the **cities should be ordered by the same criterion**. If two countries/cities have the same population, keep them **in the order in which they were entered.** Check out the examples; follow the output format strictly!

Input

- The input data should be read from the console.
- It consists of a variable number of lines and ends when the command "**report**" is received.
- The input data will always be valid and in the format described. There is no need to check it explicitly.

Examples

Input	Output
Sofia|Bulgaria|1000000 report	Bulgaria (total population: 1000000) =>Sofia: 1000000

Input	Output
Sofia|Bulgaria|1 Veliko Tarnovo|Bulgaria|2 London|UK|4 Rome|Italy|3 report	UK (total population: 4) =>London: 4 Bulgaria (total population: 3) =>Veliko Tarnovo: 2 =>Sofia: 1 Italy (total population: 3) =>Rome: 3

15. * Legendary Farming

You've beaten all the content and the last thing left to accomplish is own a legendary item. However, it's a tedious process and requires quite a bit of farming. Anyway, you are not too pretentious – any legendary will do. The possible items are:

- **Shadowmourne** – requires **250 Shards**;
- **Valanyr** – requires **250 Fragments**;
- **Dragonwrath** – requires **250 Motes**;

Shards, Fragments and **Motes** are the **key materials**, all else is **junk.** You will be given lines of input, such as **2 motes 3 ores 15 stones.** Keep track of the **key materials -** the **first** that reaches the **250 mark wins** the **race**. At that point, print the corresponding legendary obtained. Then, print the **remaining** shards, fragments, motes,

ordered by **quantity** in **descending** order, each on a new line. Finally, print the collected **junk** items, in **alphabetical** order.

Input

- Each line of input is in format **{quantity} {material} {quantity} {material} … {quantity} {material}**

Output

- On the first line, print the obtained item in format **{Legendary item} obtained!**
- On the next three lines, print the remaining key materials in descending order by quantity
  - If two key materials have the same quantity, print them in alphabetical order
- On the final several lines, print the junk items in alphabetical order
  - All materials are printed in format **{material}: {quantity}**
  - All output should be **lowercase**, except the first letter of the legendary

Examples

Input	Output
3 Motes 5 stones 5 Shards 6 leathers 255 fragments 7 Shards	Valanyr obtained! fragments: 5 shards: 5 motes: 3 leathers: 6 stones: 5
123 silver 6 shards 8 shards 5 motes 9 fangs 75 motes 103 MOTES 8 Shards 86 Motes 7 stones 19 silver	Dragonwrath obtained! shards: 22 motes: 19 fragments: 0 fangs: 9 silver: 123

16. *** Dragon Army

Heroes III is the best game ever. Everyone loves it and everyone plays it all the time. Stamat is no exclusion to this rule. His favorite units in the game are all **types** of dragons – black, red, gold, azure etc… He likes them so much that he gives them **names** and keeps logs of their **stats**: **damage, health** and **armor**. The process of aggregating all the data is quite tedious, so he would like to have a program doing it. Since he is no programmer, it's your task to help him

You need to categorize dragons by their **type**. For each dragon, identified by **name,** keep information about his **stats.** Type is **preserved** as in the order of input, but dragons are **sorted** alphabetically by name. For each type, you should also print the average **damage**, **health** and **armor** of the dragons. For each dragon, print his own stats.

There **may** be **missing** stats in the input, though. If a stat is missing you should assign it default values. Default values are as follows: health **250**, damage **45**, and armor **10**. Missing stat will be marked by **null**.

The input is in the following format **{type} {name} {damage} {health} {armor}.** Any of the integers may be assigned null value. See the examples below for better understanding of your task.

If the same dragon is added a second time, the new stats should **overwrite** the previous ones. Two dragons are considered **equal** if they match by **both** name and type.

Input

- On the first line, you are given number N -> the number of dragons to follow
- On the next N lines you are given input in the above described format. There will be single space separating each element.

Output

- Print the aggregated data on the console
- For each type, print average stats of its dragons in format **{Type}::({damage}/{health}/{armor})**
- Damage, health and armor should be rounded to two digits after the decimal separator
- For each dragon, print its stats in format **-{Name} -> damage: {damage}, health: {health}, armor: {armor}**

Examples

Input	Output
5 Red Bazgargal 100 2500 25 Black Dargonax 200 3500 18 Red Obsidion 220 2200 35 Blue Kerizsa 60 2100 20 Blue Algordox 65 1800 50	Red::(160.00/2350.00/30.00) -Bazgargal -> damage: 100, health: 2500, armor: 25 -Obsidion -> damage: 220, health: 2200, armor: 35 Black::(200.00/3500.00/18.00) -Dargonax -> damage: 200, health: 3500, armor: 18 Blue::(62.50/1950.00/35.00) -Algordox -> damage: 65, health: 1800, armor: 50 -Kerizsa -> damage: 60, health: 2100, armor: 20
4 Gold Zzazx null 1000 10 Gold Traxx 500 null 0 Gold Xaarxx 250 1000 null Gold Ardrax 100 1055 50	Gold::(223.75/826.25/17.50) -Ardrax -> damage: 100, health: 1055, armor: 50 -Traxx -> damage: 500, health: 250, armor: 0 -Xaarxx -> damage: 250, health: 1000, armor: 10 -Zzazx -> damage: 45, health: 1000, armor: 10